Rationale for a Metalevel Collaborative Design Environment

Phyo Kyaw and Cornelia Boldyreff
Department of Computer Science,
University of Durham,U.K.
phyo.kyaw@dur.ac.uk

3rd June 2003

Abstract

This paper describes a framework that provides an evolutionary approach for the development of software engineering design and the software development process in general. The approach explores a way of recording design decisions and rationale to capture how the software designs evolve during the development process by applying the concept of a matrix-based design space in a collaborative environment. It enforces the separation of concerns for various artefacts that are produced and maintained by the development team. It also presents a framework for the environment that allows developers to share artefacts and related information about the artefacts as a shared development knowledge. This paper is applicable for all software developers and organisations whose research areas are in the context of collaborative software engineering and traceability of the design.

1 Introduction

Software engineering involves a variety of artefacts, including requirement specifications, various components, documentation, development process and practise models, development team's meeting minutes, architectural models, test cases, etc. It has been well established in the literature that the *traceability* plays a major role in software development to achieve quality and maintainability [9, 3, 14]. One way to achieve traceability is to record all of the information about the artefacts that arises during their development in a systematic way, structuring it into a network of relationships that makes it intelligible to the design team and to any future developers. Such information is referred to as meta-information of the artefacts. Through the systematic structures employed, it is possible to present this information from various view points at different levels of abstraction. Currently, in most software development projects, documentation only goes as far as various specifications, designs, architectural diagrams, API descriptions and source code modules. There is no explicit recording of the design rationale behind the individual design decisions which take place during the process of producing the various documented artefacts. Most importantly decisions made during informal communication amongst developers often go unrecorded. For example, the use of eXtreme Programming (XP) practise requires all members of the team to be involved in the development of test cases and corresponding code [1]. In this case, their test cases and source code are the primary sources for recovering and understanding the design rationale.

This paper describes an collaborative environment to support software development. It is based on the concept of matrix-based design spaces [4]. This concept can be used to *record* various aspects of the design and development of software artefacts. It provides an environment to support the collaboration within the development team by presenting the relationships amongst the artefacts. Developing such a design environment is the aim of our Collaborative Determination, Elaboration and Evolution of Design Spaces (CoDEEDS) project. The main objective of this project is to provide an environment that allows developers to determine, elaborate, and evolve such design spaces, clearly identifying all of the design parameters and design constraints that are applicable in a particular development. Furthermore to facilitate constraint checking, as well as to tailor the process of

decision making for the development of software systems, the environment must incorporate knowledge support.

2 Collaborative software development and tools

The generation of software artefacts and the types of artefacts produced are dependent on the software development processes and methods employed in practice. The team may follow a detailed process model such as the Rational Unified Process (RUP) [10] and develop a number of UML models of the system, or the team may follow the well defined less formal practises of XP and produce test cases and source code. All the artefacts are updated, shared, reused, and refined as the development evolves. Collaboration amongst developers may be formal (such as formal assessment or code review meetings) as well as informal (such as coffee room chats), during the development of the various artefacts. However as the emergence of global distributed developments [6], the collaborative processes may be applied using Computer Supported Cooperative Work (CSCW) based tools, such as groupware tools, visual workspaces, and video conferencing technologies. However, these tools only provide facilities for general collaboration in an informal way, such as chat, mailing list, etc. On the other hand, there are various team-based features provided by tools such as Rational Rose and Visual Age that facilitate model and design integration and configuration management of models [16]. Figure 1 shows the five aspects of software development that the CoDEEDS framework is focused on. The main aim is not to replace nor to integrate directly with various development tools, but to capture the artefacts and the traceability information that arises during development and systematically record this with design spaces.

When an artefact, such as a conceptual model of the system, is produced, there are many different types of information is related to the artefact. They can be referred to as the meta-information of the artefacts. Some types of meta-information are design decisions and rationales for the objects inside the model, language information, process information, and system constraints. The following CoDEEDS framework presents an architecture that allows the developers to share the artefacts that they produce, record various types of meta-information, analyse the recorded meta-information, and present the related artefacts in a matrix-based multi-dimensional design space.

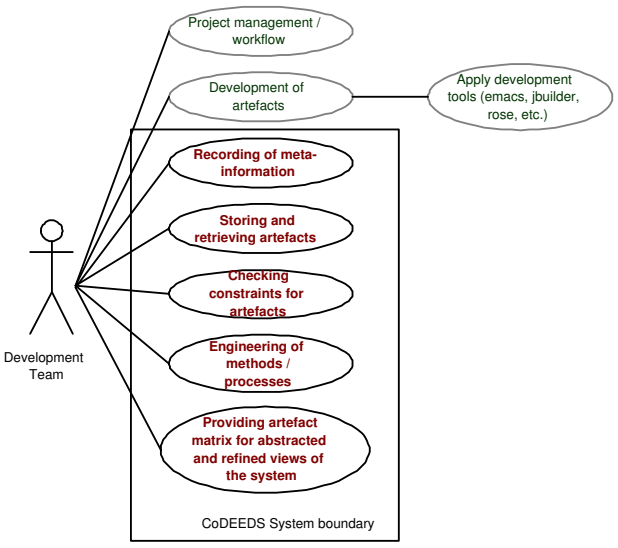

Figure 1: An overview for the use of CoDEEDS

3 CoDEEDS framework - matrix-based design space

The system uses existing heterogenous components and frameworks to provide a collaborative environment and a collection of services. Figure 2 shows an overview of the architecture of the CoDEEDS framework. The main components that form the framework are as follows.

Artefact Type Library - It contains the default type hierarchical structure for various classes of artefacts. It is used to structure the meta-information when capturing the artefacts and when presenting the artefact matrices with multi-dimensional views. The library contains a default structure of classes. However, it can be rearranged using the *artefact type configurator*. The six main base classes of the artefact type hierarchy are *development*, *system constraints*, *process*, *language*, *management* and *collaborative information*. Although the *system constraints* class can be classified under development, it is treated

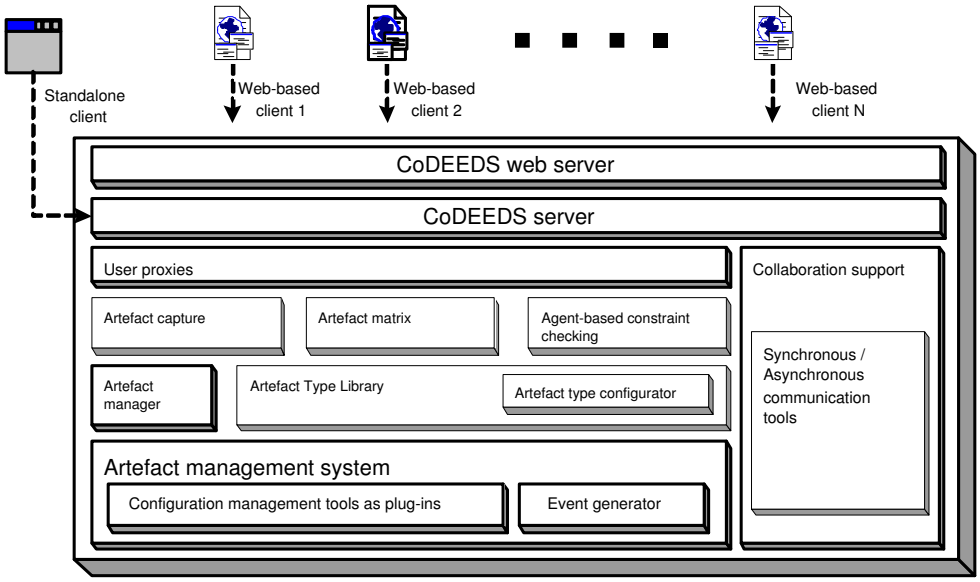

Figure 2: CoDEEDS framework overview

as a special class for performing constraint checking. As we adopted these classes from the OPEN process framework, a full list of similar classifications is presented in [8].

Artefact Capture - The Artefact Capture component allows the developer to store and retrieve artefacts and their meta-information. The recording of meta-information is done by presenting the artefact type structure from the Artefact Type Library and categorising the meta-information according to the structure. Figure 3 shows how the artefact data is wrapped with its meta-information before storing in the Artefact Management System using the Artefact Manager. The Artefact Capture component is not designed to integrated with any development, modelling nor groupware tools. However the component is designed to provide a non-invasive recording facility to instrument the collaborative software development process.

Artefact Management System - The Artefact Management System is an another core component of the framework. Currently the OSCAR

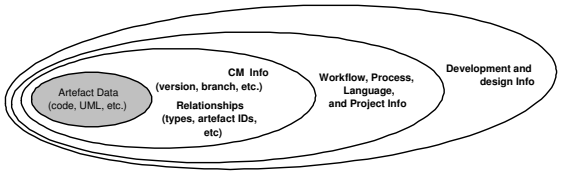

Figure 3: An overview of CoDEEDS artefact

artefact management system is applied in the framework [5]. It encapsulates different configuration management systems as plug-ins for storing artefacts. In OSCAR, an artefact is referred to as "active artefact". This means it has the awareness of its own creation, modification and can generate events to be consumed by any monitoring tasks. The OSCAR is used as an artefact repository within the CoDEEDS framework.For more information about the OSCAR, the reader is referred to [5].

Artefact Matrix - The Artefact Matrix component provides a multi-dimensional views of the artefacts related to a system. In other words, it allows the developers with different roles to view

the system with a collect level of abstraction. We adopted the concept of *design space*, which has been applied to different domain areas, including HCI and requirement engineering [2, 13, 12]. As a default view, the artefacts are categorised into three main design spaces covering the static i.e. logical view, the dynamic i.e. behavioral view, and the operational i.e. deployment view of the software system. Figure 4 shows a dimension of a matrix-based design space. Accordingly, the cell linking a particular row and column will point to another view of the matrix, showing more detailed about a particular artefact and a system constraint.

Agent-based Constraint Checking - One of the main features provided by CoDEEDS framework is semi-automatic constraint checking, if not automatic. In a software development process, an artefact, such as a conceptual model of a system, may be produced and maintained by a developer. When the model is changed, it may also change different constraints imposed to other related artefacts produced by other developers as well as the development of the system as a whole. A constraint can be a functional aspect, such as the data entry model, or it can be a system aspect (non-functional aspect), such as security model of the system.

One of the design choices to accomplish the constraint checking is to monitor the meta-information when the artefacts are created or changed. This can be done using an event monitor component. However the framework uses agent-based approach to monitor various constraints of the artefacts. The main reason for applying agent-based approach is provide an intelligent way of monitoring different aspects of the artefacts by adding rules to the agents. The agent-based monitoring is attached to the Artefact Matrix. Therefore an agent can be assigned to a particular row or column of the matrix. Similarly, the agent can also be assigned to a row, a collection of rows or a cell linking a row and a column.

The components described above are the core components of the framework. However as shown in Figure 2, there are also collaboration support and user proxy components. The former provides a collection of groupware facilities, such as mailling list, and chat. In this way, the developers may use integrated groupware facilities as well as external tools. In any case, the recording is done via the Artefact Capture component. The later is used to provide user proxy objects to each user to monitor the user's state and activities. The CoDEEDS server integrates all the above components for facilitating services to stand-alone and web-based clients. As an overview the CoDEEDS server provides a framework of matrix-based design spaces realised as a metalevel collaborative design environment. The design environment allows software developers to share information and work together to improve the quality, traceability and long-term maintainability of their software systems. The following section addresses an example to illustrate how the framework can be applied when creating software artefacts during the development.

4 A simple example to address the use of CoDEEDS framework

The following scenario describes a process of collaborative software development focusing on producing software artefacts. Figure 5 shows a simple collaborative process. At the begin of a project, the development team may perform management and process tasks, thus producing management and process *types* of artefacts. The team may then select the RUP as a process for the development and, accordingly, produce UML modelling diagrams to capture various *views* of the system.

As shown in Figure 5 (task 1), as a part of the process, a developer (A) with the architect role may produce two modelling diagrams, namely, the conceptual model and the use-case model. Then the developer may store in a datastore. To this end, there are many different types of meta-information related to these two artefacts. They can be related directly or indirectly. The meta-information includes the profiles of *modelling language*, *process*, *design*, *decisions and rationale*, *views of the system*, and *system constraints* for these artefacts. At this stage, some types of meta-information may be recorded in the development or modelling tools, some may be recorded as design documents. However, some types of meta-information such as design decisions and rationales

		Development												
		Architectural views		Requirement views				Design and analysis views		Deployment views		Implementation components views		External components views
		Conceptual model	...	Use Case model	...	...	...	Component model	...	Deployment model	...	EJB Application	...	OSCAR
Functional	Capturing artefact content	XXXXX		XXXXX				XXXXX				XXXXX		
	Capturing meta-information	XXXXX		XXXXX				XXXXX				XXXXX		
	Structuring artefact type library	XXXXX		XXXXX				XXXXX				XXXXX		
	Configuring artefact matrix	XXXXX		XXXXX				XXXXX				XXXXX		
	....													
System	Security									XXXXX		XXXXX		XXXX
	Web Service											XXXXX		
	....													

Figure 4: A sample artefact matrix

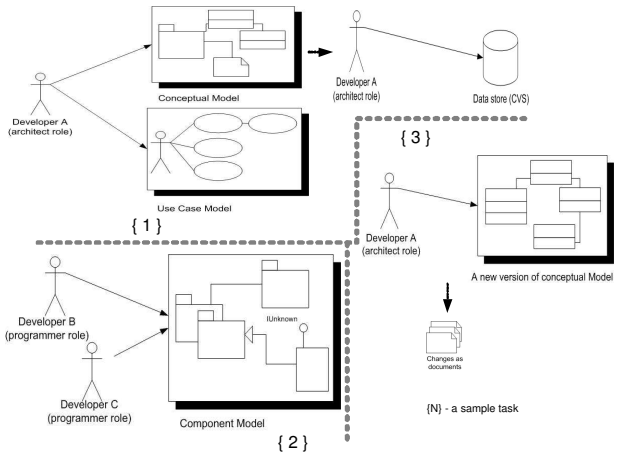

Figure 5: An overview of a sample process

may not be explicitly recorded. In such case, the CoDEEDS framework can be applied. The developer can may use the stand-alone or web-based client of the CoDEEDS framework to store the conceptual model as an artefact. The Artefact Capture component from the server side accepts the model as a file or collection of files and passes to the Artefact Management System to be stored in a configuration management system. Through the client, the Artefact Capture component presents a structure of the meta-information to be filled in, based on the data obtained from the Artefact Type Library.

As the development continues, the other two developers (B and C) with the programmer role may produce component models, as shown in Figure 5 (task 2). These component model artefacts realise all or part of the functionality addressed in the conceptual model artefacts. At this stage, these two developers require the understanding of all different types of meta-information related to the conceptual model artefact. The developers (B and C) may use the Artefact Matrix to view various aspects of the system, as shown in Figure 4.

The developers (A, B and C) may perform a collaborative process to resolve various issues about the artefacts. The collaboration may be made through meetings or by applying various groupware tools. In any case, a new type of meta-information is emerged for the artefacts. It is a collective design decisions and rationales resulting from the collaborative process. This information may be appended to the artefact's meta-information using the Artefact Capture component.

As shown in Figure 5 (task 3), the developer (A) may then change the conceptual model as the design evolves. The developer may add additional design decisions and rationale of the changes and appended to the meta-information of the artefact. Figure 6 shows a simple example of the decisions and rationale can be recorded against the system constraints. Artefact Constraint checking may monitor the events, analyse the meta-information and notify the information about the constraints to other developers.

The various types of meta-information presented above are important in two different aspects of the development. Firstly, the developers may require such meta-information as knowledge when reusing the artefacts in different domains or projects. Secondly, when artefacts are shared amongst the developers, the information provides better feedback and understand-

	Conceptual Model	
System Constraints	Decisions	Rationale
Design recording	XXXXXXXXXXXX	XXXXXXXXXX
Integration with a Artefact Management System	XXXXXXXXXXXX	XXXXXXXXXX
Configuring Artefact Matrix	XXXXXXXXXXXX	XXXXXXXXXX

Figure 6: An recording of a set of decisions and rationale against system constraints

ing of how artefacts are evolved during the development.

However currently available tools and technologies do not provide an integrated environment that shows all the meta-information presented above in a structured way. T CoDEEDS framework provides an integrated collaborative environment, which allows the developers to share artefacts and their meta-information.

We are currently implementing a framework as an open-source CoDEEDS tool. It uses various existing heterogenous components. We have reached to the stage where we can capture artefacts and their meta-information and structuring the Artefact Type Library to be presented as matrix-based design space. At this stage, we are implementing the functionalities for constraint checking and other necessary services such as groupware support and shared workspace.

5 Related Work and evaluation

The concept of design space involving the application of a matrix-based approach to record and analyse design decisions has been presented in the field of HCI [13] and software structures in[12].

As we discussed in the case study, currently available groupware tools such as [15, 7] focus on providing generic groupware features for development teams. On the other hand, there are also projects that focus on traceability and change management among all project elements, such as the approach adapted in the OPHELIA project [11]. Their approach focuses on integrating products resulting from various development tools and performing traceability as well as automatic notifications about changes to the products. However, our main concern is providing high-level collaborative support by recording meta-information and presenting it with artefact matrices. When designing a new application, the determination of relevant design methods, design parameters and constraints is important to all members of the development team. In this paper, we have presented a way of recording such important information in a collaborative environment. The automatic constraint checking can also be performed. However, since we do not analyse the actual data content of the artefacts, the richness of the meta-information is limited to the data provided by the developers and the tools. The CoDEEDS framework do not provide facilities to directly apply notes or documents inside the development or modelling tool as the meta-information, as it does not integrate with any development tool. However it provides a structure to import information from various tools.

One of the open issues of tool-based development, such as the CoDEEDS, is the lack of hard evidence for success. As with many other projects that are based on providing tool support, detailed evaluation of the tool can only be made after the release of the tool. We are planning to release the first version of the tool in very near future.

6 Conclusion

As an overview, we have described a collaborative environment with a design recording facility to assist software development teams engaged in collaborative software development. The framework also addresses the use of various views that separate various concerns during the design. The research of the CoDEEDS project is to provide a closer relationship amongst the CSCW-based research on collaboration environment, traceability of the design, and evolution of design spaces.

References

[1] Ambler, S. W. *Agile modeling.* Wiley, New York, 2002.

[2] Baum, L., Becker, M., Geyer, L., and Molter, G. Mapping requirements to reusable components using design spaces. In *ICRE* (2000), pp. 159–167.

[3] Boldyreff, C., Burd, E., Hather, R., Munro, M., and Younger, E. Greater under-

standing through maintainer driven traceabilit. *IEEE Computer Press* (1996), 100–106.

[4] Boldyreff, C., Kyaw, P., Nutter, D., and Rank, S. Architectural framework for a collaborative design environment. In *Proceedings of Second ASERC Workshop on Software Architecture* (Banff, Canada, 2003).

[5] Boldyreff, C., Nutter, D., and Rank, S. Architectural requirements for an open source component and artefact repository system within GENESIS. In *Proceedings of the Open Source Software Development Workshop* (Newcastle, UK, Feb. 2002), C. Gacek and B. Arief, Eds., pp. 176–196.

[6] Braun, A., Dutoit, A. H., and Brugge, B. A software architecture for knowledge acquisition and retrieval for global distributed teams. In *Proceedings of the 3rd International Workshop on Global Software Development* (Portland, Oregon, 2003).

[7] Fanderclai, T. L. Community building in CVW. *ACM SIGGROUP Bulletin 19*, 3 (1998), 18–21.

[8] Firesmith, D. G., and Henderson-Sellers, B. *The OPEN Process Framework* The OPEN Series. Addison Wesley, 2002.

[9] Gotel, O., and Finkelstein, A. W. An analysis of the requirements tracebility problem. In *Proceedings of the International Conference on Requirements Engineering* (Colorado Springs, Colorado, 1994).

[10] Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process* Object Technology Series. Addison-Wesley, 1999.

[11] Kowalczykiewicz, K., and Weiss, D. Traceability: Taming uncontrolled change in software development, 2002.

[12] Lane, T. G. Studying software architecture through design spaces and rules. Tech. Rep. CMU/SEI-90-TR-18, November 1990.

[13] Maclean, A., and McKerlie, D. Design space analysis and use representations. In *Scenario-based design: envisioning work and technology in system development* (1995), John Wiley and Sons, Inc., pp. 183–207.

[14] Olsson, T., and Grundy, J. Supporting traceability and inconsistency management between software artefacts. In *Proceedings of the 2002 IASTED International Conference on Software Engineering and Applications* (Boston, USA, 2002).

[15] phpGroupware. phpgroupware home page, 2002. Avaliable at `www.phpgroupware.org`.

[16] Rational Rose. Rose visual modelling tool, 2002. Avaliable at `www.rational.com`.